Profiling OGSA-DAI Performance
for Common Use Patterns

Bartosz Dobrzelecki[1], Mario Antonioletti[1], Jennifer M. Schopf[2,3], Alastair C. Hume[1], Malcolm Atkinson[2], Neil P. Chue Hong[1], Mike Jackson[1], Kostas Karasavvas[2], Amy Krause[1], Mark Parsons[1], Tom Sugden[1], and Elias Theocharopoulos[2]

1. EPCC, University of Edinburgh, JCMB, The King's Buildings, Mayfield Road, Edinburgh EH9 3JZ, UK.
2. National e-Science Centre, University of Edinburgh, Edinburgh EH8 9AA, UK.
3. Distributed Systems Laboratory, Argonne National Laboratory, Argonne, IL, 60439 USA.

Abstract

OGSA-DAI provides an extensible Web service-based framework that allows data resources to be incorporated into Grid fabrics. The current OGSA-DAI release (OGSA-DAI WSI/WSRF v2.2) has implemented a set of optimizations identified through the examination of common OGSA-DAI use patterns. In this paper we describe these patterns and detail the optimizations that have been made for the current release based on the profiles obtained. These optimizations include improvements to the performance of various data format conversion routines, the introduction of more compact data delivery formats, and the adoption of SOAP with attachments for data delivery. We quantify the performance improvements in comparison to the previous OGSA-DAI release.

1. Introduction

The *Open Grid Services Architecture – Data Access and Integration* (OGSA-DAI) project [OD] aims to provide the e-Science community with a middleware solution to assist with the access and integration of data for applications working within Grids. Early Grid applications focused principally on the storage, replication, and movement of file-based data, but many of today's applications need full integration of database technologies with Grid middleware. Not only do many Grid applications already use databases for managing metadata, but increasingly many are associated with large databases of domain-specific information, for example, biological or astronomical data.

OGSA-DAI offers a collection of services for adding database access and integration capabilities to the core capabilities of service-oriented Grids, thus allowing structured data resources to be integrated with Grid applications. A systematic performance analysis of OGSA-DAI v2.1 [AAB+05] led to the emphasis on performance for the v2.2 release.

Our current work has focused on identifying common use patterns and their associated bottlenecks and then improving the performance of these particular use cases. The optimizations implemented include speeding data format conversion routines, introducing more compact data delivery formats, and using SOAP with attachments for data delivery. Section 2 references more detailed descriptions of OGSA-DAI and outlines related middleware and performance studies. Section 3 describes the use patterns adopted to identify the performance bottlenecks and describes the performance enhancements made. Section 4 quantifies the improvements resulting from these changes, comparing the performance of the present WSRF OGSA-DAI release v2.2, with the previous v2.1 release. Section 5 presents some conclusions derived from this work.

2. Related Work

Various publications provide information about the design of OGSA-DAI [AAB+05a], ways in which it can be used [KAA+05], the related WS-DAI family of specifications [AKP+06], and details about the OGSA-DQP software [AMP+03] which adds distributed query-processing capabilities through OGSA-DAI. Up-to-date documentation, tutorials, and

downloads related to OGSA-DAI are available from [OD].

2.1. Related Applications

Three current projects have closely related functionality to that of OGSA-DAI: the Storage Resource Broker, WebSphere Information Integrator, and Mobius.

The Storage Resource Broker (SRB) [SRB], developed by the San Diego Supercomputer Center, provides access to *collections*, or sets of data objects, using attributes or logical names rather than their physical names or locations. SRB is primarily file oriented but can also work with data objects, including archival systems, binary large objects in a database management system, database objects that may be queried by using SQL, and tape library systems. By contrast, OGSA-DAI takes a database oriented approach which also includes access to files. These two approaches are generally suited to differing problems, but SRB and OGSA-DAI can complement each other.

WebSphere Information Integrator (WSII), a commercial product from IBM, is commonly used to search data spanning organisational domains, data federation and replication, data transformation, and data event publishing [WSII]. Data federation allows multiple data sources to be queried and accessed through a single access point. IBM recently developed a Grid wrapper for WSII using OGSA-DAI, taking advantage of its abstraction capabilities, to wrap additional data resources that WSII can then access [LMD+05]. A more detailed comparison between OGSA-DAI and WSII can be found in [SH05].

Mobius [Mob], developed at Ohio State University, provides a set of tools and services to facilitate the management and sharing of data and metadata in a Grid. In order to expose a resource in Mobius, the resource must be described by using an XML Schema, which is then shared via their Global Model Exchange. The resource can then be accessed by querying the Schema using, for example, XPath. OGSA-DAI, in contrast, does not require an XML Schema to be created for a resource; rather, it directly exposes that information (data and metadata/schema) and is queried by using the resource's intrinsic querying mechanisms.

Several other projects, including ELDAS [ELD] and Spitfire [SF], also address the use of databases in a Grid environment but are not as commonly used.

2.2. Related Performance Studies

The Extreme Grid Web Services group at Indiana University have examined the performance of SOAP for high-performance and Grid computing [CGB02, GSC+00]. They have developed an *XML Pull Parser* implementation that is significantly faster than Xerces [Slo04]. This work concentrates on the XML processing and is not specific to database use patterns but one that potentially could be exploited by OGSA-DAI through its use of Web services.

A large body of work exists on benchmarking relational database systems. In particular, the Wisconsin Benchmark [Gra93] consists of queries that test the performance of small numbers of join operations. The XML Benchmark Project [SWK+01] has presented an approach to benchmarking XML databases [SWK+01a]. However, none of this work has looked at benchmarking Web service interfaces to databases – the area that OGSA-DAI occupies.

Some attempts have been made to use standard benchmarks to investigate the overheads of providing database access through Web service-style interfaces, for example by using TPC-H [HIM02]. This particular work focuses on network and encryption overhead but using a non SOAP-based interface. Nevertheless, this work is of interest for comparison for SOAP-based access.

Previous work has also been done to try to understand the performance of earlier versions of OGSA-DAI [JAC+03, AAB+05, AGM+05]. This work mainly looked at particular technical issues, whereas the work presented in this paper seeks to use common use patterns as a starting point for on-going optimisation.

3. Performance Bottlenecks in Common Use Patterns

OGSA-DAI employs Web services to expose intrinsic data resource capabilities and the data contained to its clients. In most instances, the data resources used are relational databases that require read-only access (query) and no write access (update/insert). For this reason, our study has focused on relational databases and used the execution of an SQL query as the base test case.

3.1. Use Case 1: Executing an SQL Query on a Remote Server

A typical OGSA-DAI client-service interaction involves a client running an SQL query through a remote OGSA-DAI service that then returns the query response, typically some data, in an XML document. This interaction involves the following six steps:

(1) The client sends a request containing the SQL query in a SOAP message to an OGSA-DAI service.
(2) The server extracts the request from the SOAP message, and the SQL query is executed on the relational database.
(3) The query results are returned from the relational database to the OGSA-DAI server as a set of Java ResultSet objects.
(4) The server converts the Java ResultSet objects into a format suitable for transmission back to the client, such as WebRowSet.
(5) This data is sent back to the client in a SOAP message.
(6) The client receives the SOAP message, unpacks the data, and converts it back to a ResultSet object (assuming this is a Java client).

3.1.1. Improvement 1: Faster Conversion

Profiling this use case showed that the conversion process, where a ResultSet object is converted to the WebRowSet format on the server (step 4), as well as the inverse process on the client (step 6), was the primary performance bottleneck, and an obvious area for improvement.

Previously, these converter routines were also applied to produce binary data, going from ResultSets to some suitable binary format. However, the cost of iteratively having to convert ResultSets to binary data proved to be too high. So, the converters were restricted to only deal with text based formats. This benefited the performance. In addition, a routine that used a regular expression Java API to escape XML special characters in data fields proved to be too expensive and was thus replaced by a more efficient parser that worked on arrays. These combined modifications improved the performance of the converters.

3.1.2. Improvement 2: Change in Data Format

In analyzing the overhead for our first use case, we also noticed that using WebRowSet as an intermediate delivery format added a significant amount of XML mark-up that increased the amount of data that needed to be transferred between the client and server, often up to twice the original size. In addition, the parsing of the messages out of XML could be slow.

This scenario also identified the fact that WebRowSet was only being used as an intermediate delivery format and was thus possibly incurring an unnecessary overhead. Hence, a second improvement we investigated was the use of an alternative intermediate delivery format, namely, *Comma Separated Values* (CSV). This format uses space more efficiently and is easier to parse, but it has two significant drawbacks. First, it provides only limited support for metadata – namely, an optional line with column names – so the embedding of metadata has weaker support than is the case with the WebRowSet format. Second, as there is no standard for representing relational data in CSV format, third-party tools may have difficulty interpreting OGSA-DAI-generated CSV files, despite

the fact that common conventions are used and they are internally consistent.

The reduction in data size in going from a WebRowSet to a CSV format can be estimated by calculating the space required to represent the same result in each format. This can be done by calculating the number of extra characters needed to describe a row of data. Assuming for CVS data that all fields are wrapped in double quotes and that there are no escaped characters, then the extra number of characters needed to represent a row in CSV document is: (number_of_columns * 3) – two quotes and a comma – whereas for WebRowSet the use of specific WebRowSet defined XML tags make this number: (number_of_columns * 27)+25. So, WebRowSet always requires at least nine time as many non-data characters as CSV.

3.2. Use Case 2: Transferring Binary Data

A slight variation on the previous use case involves using OGSA-DAI to provide access to files stored on a server's file system. Commonly, these could be large binary data files (e.g., medical images), stored in a file system, with the associated metadata for these files stored separately in a relational database. The client queries the databases to locate any files of interest, which are then retrieved by using the OGSA-DAI delivery mechanisms. Files are retrieved separately from the SOAP interactions for data transport efficiency. In some cases, however, it can be more convenient for a client to receive the data back in a SOAP response message rather than using an alternative delivery mechanism.

The implementation of this scenario includes the same six steps as in the first use case except that step 4 now requires the conversion of a binary file to a text-based format, usually Base64 encoding, in order for it to be sent back in a SOAP message, and step 6 includes decoding of the file back into its original binary format.

3.2.1. Improvement 3: SOAP with Attachments

The major bottleneck arising from this scenario is the Base64 encoding of the binary data for inclusion in a SOAP message. This encoding requires additional computation at both the client and the server side. Moreover, the converted data is approximately 135% of the size of the original file, clearly impacting the efficiency of the data transfer.

We have addressed both of these concerns by using SOAP messages with attachments [BTN00]. This approach significantly reduces the time required to process SOAP messages and allows the transfer of binary data to take place without necessitating Base64 encoding. The one difficulty with this approach is that, as SOAP messages with attachments are not a standard feature of all SOAP specifications, interoperability issues can arise.

4. Experimental Results

To quantify the effect of the performance improvements outlined in the preceding section, we compared the performance of OGSA-DAI WSRF v2.2 with OGSA-DAI WSRF v2.1.

4.1. Experimental Setup

We ran our experiments using an Apache Tomcat 5.0.28 / Globus Toolkit WS-Core 4.0.1 stack. Our experimental setup consisted of a client machine and a server machine on the same LAN. We ran the server code on a Sun Fire V240 Server, which has a dual 1.5 GHz UltraSPARC IIIi processor and 8 GB of memory, running Solaris 10 with J2SE 1.4.2_05. The client machine was a dual 2.40GHz Intel Xeon system running Red Hat 9 Linux with the 2.4.21 kernel and J2SE 1.4.2_08. Both the client JVM and the JVM running the Tomcat container were started in server mode by using the `-server -Xms256m -Xmx256m` set of flags.

The network packets in these experiments traversed two routers, and iperf 1.7.0 found the network bandwidth to be approximately 94 Mbits/s. The average round-trip latency was less than 1 ms.

The database used in the experiments was MySQL 5.0.15 with MySQL Connector/J driver version 3.1.10. We used *littleblackbook*, the sample database table distributed with OGSA-DAI, for all experiments. The average row length for this table is 66 bytes. The rows have the following schema: int(11), varchar(64), varchar(128), varchar(20).

Before we took any measurements, both the client and server JVMs were warmed up. Then each test was executed 10 times. The results reported are the average of these runs, with error bars indicating +/- standard deviation.

4.2. Faster Conversions and Change in Data Format

Our first set of experiments was based on the two improvements suggested by our first use case, namely, optimizing the code to do the data format conversions faster and evaluating the use of CSV instead of WebRowSet for an intermediate format.

For a set of queries, returning results consisting from 32 to 16,384 rows, we measured the time to perform the 6 steps involved in a client-service interaction, including the translation of the results into (and out of) WebRowSet or CSV formats as appropriate.

Figure 1 shows the overall timing results. Queries for 512 rows or greater show a significant improvement, up to 35% by simply optimizing the WebRowSet conversion (Improvement 1). The use of CSV instead of WebRowSet (Improvement 2) also shows a significant improvement for larger queries, up to 65% over the original v2.1 and about 50% over simply optimizing the conversion.

Figure 2 shows in more detail the performance improvements on the server side using Apache Axis logging to obtain the times spent in the different phases of the SOAP request processing. We divide the server performance into three phases:

1. Axis Parsing: the time spent in Apache Axis parsing a SOAP request.

2. OGSA-DAI Server: the time OGSA-DAI spent performing the requested activities and building the response document.

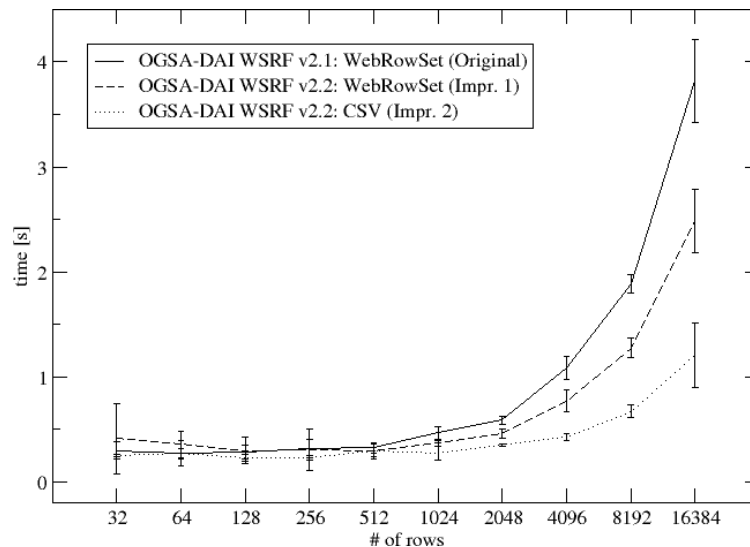**3. Message Transfer**: the time the server spent sending a message back to the client.

Figure 1: Measurements comparing the effects of better conversion code (Improvement 1) and using CSV formatting instead of WebRowSet format (Improvement 2) against the original v2.1 code. Results include both client and server times.
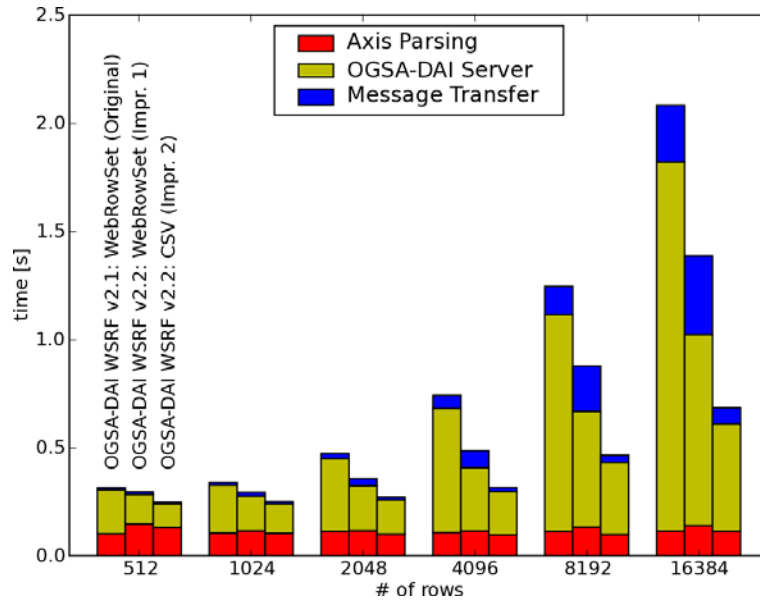
Figure 2: Time spent in the server only, split into three phases: Apache Axis parsing, OGSA-DAI server work, and the message transfer to the client.

The time spent in the Axis parsing phase is roughly constant as we always perform the same operation. In all cases the time spent in the OGSA-DAI server phase dominates and generally increases systematically with the size of the query results obtained. The largest portion of this phase is spent translating the ResultSet objects from a Java ResultSet object into WebRowSet or CSV. The optimised conversion to WebRowSet (Improvement 1) works up to 50% faster than the original version for large result sets. By using CSV instead of WebRowSet (Improvement 2), we also see a large reduction in delivery time and reduced network traffic, the Message Transfer phase.

4.3. Using SOAP Attachments

The second set of experiments we ran was to test the use of SOAP with attachments, as outlined in our second use case and Improvement 3 in Section 3.2. We compare using Byte64 encoding and returning the data in the body of a SOAP message to using SOAP messages with attachments to transfer a binary file.

Figure 3 shows the time taken to transfer binary data of increasing size using both delivery methods. We only have data for the Original v2.1 code up to 8MB file sizes because the process of Base64 encoding and building SOAP response for file sizes of 16MB upwards consumed all the heap memory available to the JVM and consequently caused the JVM to terminate. The performance gain shows nonlinear growth with increasing file size. Transferring an 8 MB file as a SOAP attachment takes only 25% of the time needed to transfer the same file inside the body of a SOAP message. This improvement is due to the fact that SOAP attachments do not need any special encoding, and less time is spent processing XML because the data is outside the body of the SOAP message.

Figure 4 gives additional detail about the server-side performance using the three previously defined phases. The Axis parsing phase is roughly constant, as before. During the OGSA-DAI server phase, the original SOAP delivery method is CPU bound because of the Base64 conversion required, while the performance of the new SOAP with attachments case is generally much better, limited mainly by the performance of the I/O operations rather than those of the CPU.
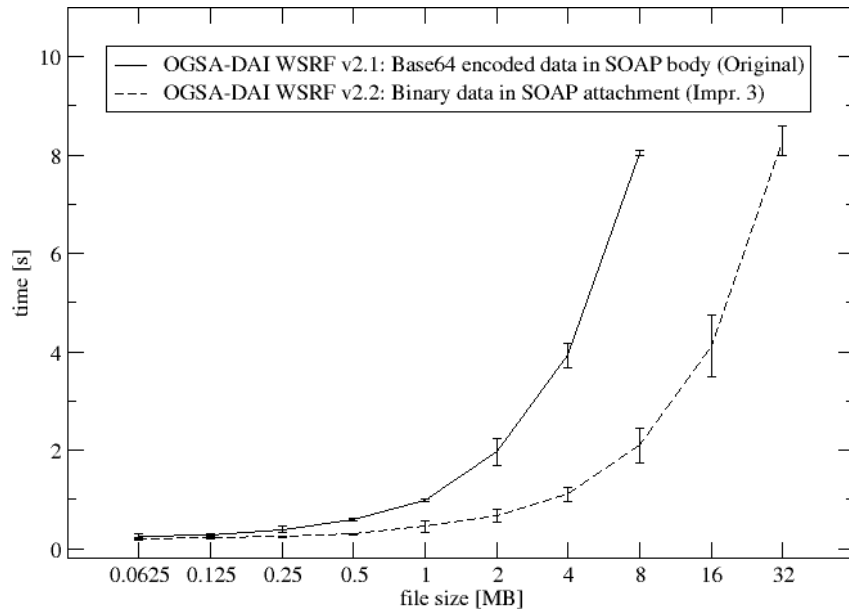
Figure 3: Time taken to transfer a binary file using Base64 encoded data inside the body of a SOAP message and as a SOAP attachment (Improvement 3).
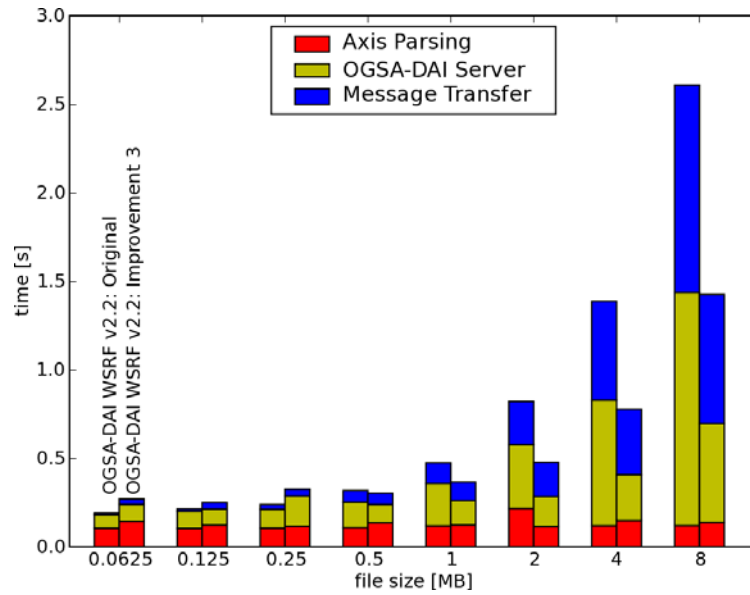

Figure 4: Time spent on server side split into phases. For each group the bar on the left measures the time spent sending binary data inside a SOAP message (Original) while the right bar corresponds to the approach where binary data is sent as a SOAP attachment (Improvement 3).

A similar performance gain is seen in the message transfer phase, where the absence of Base64 encoding reduces the quantity of data that needs to be transferred by up to 35% for the SOAP with attachments case.

4.4. SQL Results Delivery Using SOAP Attachments

The final experiment combined the previous two, by repeating the SQL query results retrieval from the first experiment but also using SOAP with attachments for data delivery. Figure 5 shows that there is little difference in the performance for
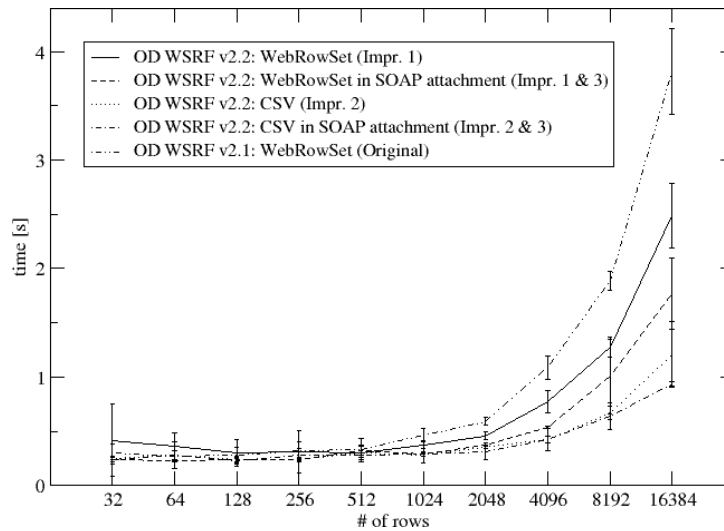
Figure 5: Execution time for scenarios fetching SQL results converted to XML and CSV data using two delivery mechanisms: delivery inside the body of a SOAP message and delivery as a SOAP attachment.

small data sets (less than 512 rows). For larger data sets, however, SOAP with attachments can achieve transfer times that are up to 30% faster than when using WebRowSet for delivery. Only the largest data set sees a performance gain when CSV formatting is used with SOAP attachments.

5. Conclusions

This paper summarises part of an ongoing effort to improve the performance of OGSA-DAI. We have analysed two typical use patterns, which were then profiled and the results used as a basis for implementing a focused set of performance improvements. The benefit of these has been demonstrated by comparing the performance of the current release of OGSA-DAI, which includes the performance improvements, with the previous release, which does not. We have seen performance improvements of over 50% in some instances. Source code and the results data are available from [Dob06].

Acknowledgements

References

[AAB+05] M Antonioletti, M. Atkinson, R. Baxter, A Borley, N. P. Chue Hong, P. Dantressangle, A. C. Hume, M. Jackson, A. Krause, S. Laws, M. Parsons, N. W. Paton, J. M. Schopf, T. Sugden, P. Watson, and D. Vyvyan, OGSA-DAI Status and Benchmarks, Proceedings of the UK e-Science All Hands Meeting, 2005.

[AAB+05a] M. Antonioletti, M.P. Atkinson, R. Baxter, A. Borley, N.P. Chue Hong, B. Collins, N. Hardman, A. Hume, A. Knox, M. Jackson, A. Krause, S. Laws, J. Magowan, N.W. Paton, D. Pearson, T. Sugden, P. Watson, and M. Westhead. The Design and Implementation of Grid Database Services in OGSA-DAI. Concurrency and Computation:

Practice and Experience, Volume 17, Issue 2-4, Pages 357-376, February 2005.

[AGM+05] M.N. Alpdemir, A. Gounaris, A. Mukherjee, D. Fitzgerald, N. W. Paton, P. Watson, R. Sakellariou, A. A.A. Fernandes, and J. Smith, Experience on Performance Evaluation with OGSA-DQP, Proceedings of the UK e-Science All Hands Meeting, 2005.

[AKP+06] M. Antonioletti, A. Krause, N. W. Paton, A. Eisenberg, S. Laws, S. Malaika, J. Melton, and D. Pearson. The WS-DAI Family of Specifications for Web Service Data Access and Integration. ACM SIGMOD Record, Vol 35, No 1, pp48-55, 2006.

[AMP+03] M. N. Alpdemir, A. Mukherjee, N. W. Paton, P. Watson, A. A. A. Fernandes, A. Gounaris, and J. Smith. Service-Based Distributed Querying on the Grid. Service-Oriented Computing - ICSOC 2003 Editors: M. E. Orlowska, S. Weerawarana, M. P. Papazoglou, J. Yang. Lecture Notes in Computer Science, Volume 2910, pp. 467-482 Springer Berlin/Heidelberg 2003.

[BTN00] J. J Barton, S. Thatte, and H. F. Nielsen. *SOAP Messages with Attachments*. W3C Note 11 December 2000.

[CGB02] K. Chiu, M. Govindaraju, and R. Bramley, Investigating the Limits of SOAP Performance for Scientific Computing, Proceedings of HPDC 2002.

[Dob06] B. Dobrzelecki, Code and raw data from experiments, www.ogsadai.org.uk/documentation/scenarios/performance, 2006.

[ELD] ELDAS (Enterprise Level Data Access Services), EDIKT, www.edikt.org/eldas.

[Gra93] J. Gray, Database and Transaction Processing Performance Handbook. www.benchmarkresources.com/handbook, 1993.

[GSC+00] M. Govindaraju, A. Slominski, V. Choppella, R. Bramley, and D. Gannon, On the Performance of Remote Method Invocation for Large-Scale Scientific Applications, Proceedings of SC'00, 2000.

[HIM02] H. Hacigumus, B. Iyer, and S. Mehrotra, Providing database as a service, Proceedings of 18th International Conference on Data Engineering 2002.

[JAC+03] M. Jackson, M. Antonioletti, N. Chue Hong, A. Hume, A. Krause, T. Sugden, and M. Westhead, Performance Analysis of the OGSA-DAI Software, Proceedings of the UK e-Science All Hands Meeting, 2003.

[KAA+05] K. Karasavvas, M. Antonioletti, M.P. Atkinson, N.P. Chue Hong, T. Sugden, A.C. Hume, M. Jackson, A. Krause, and C. Palansuriya. Introduction to OGSA-DAI Services. Lecture Notes in Computer Science, Volume 3458, Pages 1-12, May 2005.

[LMD+05] A. Lee, J. Magowan, P. Dantressangle, and F. Bannwart. Bridging the Integration Gap, Part 1: Federating Grid Data. IBM Developer Works, August 2005.

[Mob] Mobius, projectmobius.osu.edu.

[OD] Open Grid Services Architecture – Data Access and Integration (OGSA-DAI), www.ogsadai.org.uk.

[SH05] R. O. Sinnott and D. Houghton, Comparison of Data Access and Integration Technologies in the Life Science Domain, Proceedings of the UK e-Science All Hands Meeting 2005, September 2005.

[SF] Spitfire, edg-wp2.web.cern.ch/edg-wp2/spitfire.

[Slo04] A. Slominski. www.extreme.indiana.edu/~aslom/xpp_sax2bench/results.html, 2004.

[SRB] Storage Resource Broker (SRB), www.sdsc.edu/srb.

[SWK+01] A. R. Schmidt, Florian Waas, M. L. Kersten, D. Florescu, I. Manolescu, M. J. Carey, and R. Busse, The XML Benchmark Project, CWI (Centre for Mathematics and Computer

Science), Amsterdam, The Netherlands, 2001.

[SWK+01a] A. Schmidt, F. Waas, M. Kersten, D. Florescu, M. J. Carey, I. Manolescu, and R. Busse, Why and How to Benchmark XML Databases, ACM SIGMOD Record Volume 30, Issue 3, Pages 27-32, September 2001.

[WSII] Web Sphere Information Integrator (WSII), www.ibm.com/software/data/integration.

[WRS] WebRowSet XML Schema definition, java.sun.com/xml/ns/jdbc/webrowset.xsd.